TRANSFORMERS: Robust Spatial Joins on Non-Uniform Data Distributions

Mirjana Pavlovic[†], Thomas Heinis[¶*], Farhan Tauheed[§*], Panagiotis Karras[‡], Anastasia Ailamaki[†]

[†]*École Polytechnique Fédérale de Lausanne, Switzerland*

[¶]*Imperial College London, United Kingdom*

[§]*Oracle Labs Zurich, Switzerland*

[‡]*Skolkovo Institute of Science and Technology, Russia*

***Abstract*—Spatial joins are becoming increasingly ubiquitous in many applications, particularly in the scientific domain. While several approaches have been proposed for joining spatial datasets, each of them has a strength for a particular type of density ratio among the joined datasets. More generally, no single proposed method can efficiently join two spatial datasets in a *robust* manner with respect to their data distributions. Some approaches do well for datasets with contrasting densities while others do better with similar densities. None of them does well when the datasets have locally divergent data distributions.**

In this paper we develop TRANSFORMERS, an efficient and robust spatial join approach that is indifferent to such variations of distribution among the joined data. TRANSFORMERS achieves this feat by departing from the state-of-the-art through adapting the join strategy and data layout to local density variations among the joined data. It employs a join method based on data-oriented partitioning when joining areas of substantially different local densities, whereas it uses big partitions (as in space-oriented partitioning) when the densities are similar, while seamlessly switching among these two strategies at runtime. We experimentally demonstrate that TRANSFORMERS outperforms state-of-the-art approaches by a factor of between 2 and 8.

I. Introduction

In many different applications, the efficient execution of spatial joins becomes increasingly important. In scientific applications, for example, spatial joins are used to determine the location of synapses in brain models [1], in medical imaging to determine proximity of cells and in geographical information systems spatial joins detect collisions between geographical features like houses, roads, etc.

Given the importance of the application, several methods have been developed to perform disk-based spatial joins [2], [3]. Methods developed in the past can efficiently join two or more disk-based spatial datasets of uniform distribution of element locations. They are, however, inadequate when joining two spatial datasets, each with a skewed distribution of element location [4]. Doing so, however, is important as the datasets joined rarely have a similar distribution.

Formally, the goal is to develop a robust and efficient method to spatially join two disk-based datasets, each with a non-uniform distribution of element location, i.e., with *locally varying* densities. More precisely, each dataset D can have areas d_i with a considerable difference in density such that $\forall i, j$ with $i \neq j$ $|d_i| << |d_j|$ or $|d_i| >> |d_j|$ along with areas of similar density, $|d_i| \approx |d_j|$. When joining such datasets A and B we have to efficiently join areas $a_i \in A$ and $b_i \in B$ with similar spatial extent and location regardless of their density.

*This work was done while the author was at EPFL.

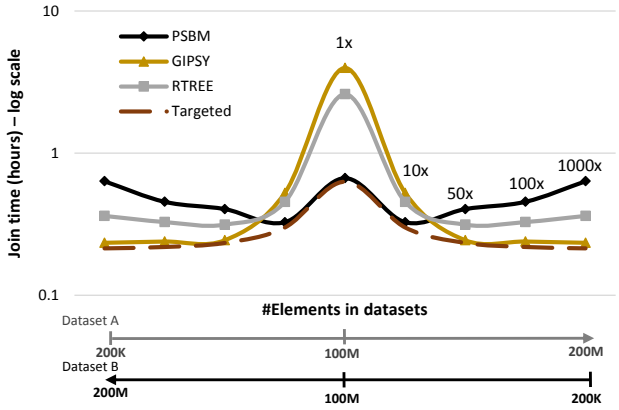

Fig. 1. Join time for datasets with variable relative density.

As we show with an experiment in Figure 1, no previously proposed approach achieves to join all combinations of density ratios of a_i and b_i, e.g., $|a_i| \approx |b_i|$, $|a_i| << |b_i|$ or $|b_i| << |a_i|$, in a *robust* manner. In this experiment we join different combinations of datasets A and B and measure the join time. More precisely, we steadily increase the density of dataset A (starting from 200K spatial elements) and decrease it for B (starting from 200M) and measure the execution time for joining each combination. The two datasets represent areas with different or similar density. The numbers above the curve indicate the ratio of density between the datasets; a more detailed explanation is given in Section II-A.

As the experimental results in Figure 1 show, none of the existing approaches performs best in every situation. Approaches based on space-oriented partitioning (e.g., PBSM [3]) do well when joining datasets of similar density while data-oriented partitioning approaches (e.g., based on the R-Tree [2], [5] and particularly GIPSY [4]) are more efficient when joining datasets with contrasting density. None of the existing approaches joins the datasets (or areas) with robust performance across different density ratios.

In this paper we thus develop TRANSFORMERS, a novel disk-based join approach that handles this robustness problem with respect to local density variations and targets to achieve performance as is shown in Figure 1 (Targeted). For all areas $a_i \in A$ and $b_i \in B$, TRANSFORMERS decides *locally* which area is dense and which is sparse and adapts the join strategy as well as the data layout accordingly. With its adaptive join strategy, TRANSFORMERS achieves a more robust join performance across different density ratios and outperforms

previous work by a factor of between 2 and 8.

Our contributions are as follows:

- We show that static strategies in the join phase lead to sub-optimal, non-robust performance when joining datasets with non-uniform distributions.
- We develop TRANSFORMERS, a novel approach that detects local variations in distributions and adapts its strategy and the data layout on the fly accordingly.
- We demonstrate robustness as well as substantial performance improvements achieved with TRANSFORMERS on scientific and synthetic datasets.

The remainder of this paper is organized as follows. In Section II we motivate TRANSFORMERS with an initial set of measurements confirming our assumptions. We give an overview of our approach in Section III and then discuss in detail the indexing process in Section IV, the join process in Section V as well as transformations in Section VI. In Section VII we demonstrate the performance of TRANSFORMERS. We review related work on disk-based spatial join methods in Section VIII and draw conclusions in Section IX.

II. Motivation

Spatial joins have become a crucial operation across different scientific and business applications. Frequently the two datasets to be joined have a considerably different density and data distribution. For example, while dataset A has a uniform distribution and density, the join performance may be affected by the local variation in distribution and density of dataset B.

In Figure 2 we illustrate several examples of local variations in distribution and density. *Uniform* (left) illustrates two datasets with similar distribution and density throughout the area they cover. In case of *contrasting density* (middle) both datasets have similar distribution, however, skew is introduced through the different number of elements in the datasets. The datasets in *contrasting distribution* (right), on the other hand, have a similar number of elements but a different distribution. In the latter two cases, *contrasting density* and *contrasting distribution*, a join between two datasets will join areas with considerably different densities. As we illustrate with the following experiments, skew due to these variations in density leads to significant overhead, i.e., unnecessary processing.

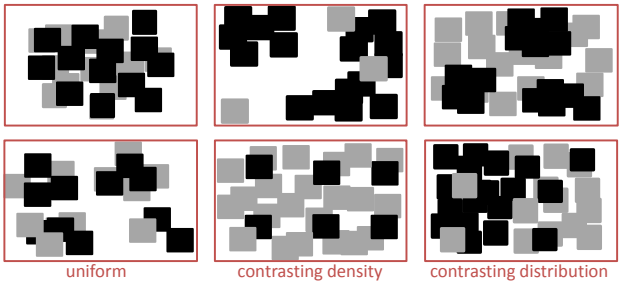

Fig. 2. Illustration of variations in distribution and density. Each figure shows two datasets, one with grey elements and the other with black ones.

A. Motivating Experiment

We illustrate the shortcoming of the state of the art with experiments where we join datasets with contrasting densities. To achieve an approximation of joining two disk-based datasets that differ significantly in local densities, we join nine pairs of datasets with uniform data distribution whose density ratio ($|a|/|b|$) varies between 10^{-3} and 1000 (numbers shown above the curves in Figure 1). To obtain nine pairs of datasets with contrasting densities, we increase the density of one dataset (starting from 200K elements) and decrease it for the other (starting from 200M) in consecutive steps.

We measure the execution time (without taking into account the indexing phase) for the join for the fastest and most broadly used disk-based spatial join methods, i.e., PBSM [3], the Synchronized R-Tree (R-Tree [2]) and GIPSY [4]. We use the best configuration for each approach, e.g., the number of partitions/tiles for PBSM, page size and fanout for the R-Tree. The results are shown in Figure 1.

When joining a sparse area $a_i \in A$ with a dense area $b_i \in B$, only a very small subset needs to be retrieved from b_i (and tested against a_i). Approaches based on space-oriented partitioning like PBSM [3], however, read considerably more data than is required from b_i and thus also require more comparisons. Due to coarse-grained partitioning inherent in these methods, almost all of b_i is read for the join, leading to excessive disk accesses and comparisons (points 1000x, 100x, 50x). Space-oriented partitioning methods, on the other hand, are efficient when joining areas of similar density (point 1x).

Data-oriented partitioning approaches (based on the R-Tree [6] or others, e.g., the synchronized R-Tree [2]) use a very fine-grained partitioning that enables them to retrieve data very selectively. As Figure 1 shows, doing so proves efficient on contrasting densities but their inherent problem of structural overlap leads them to read and test more data than necessary, making them comparatively slow when joining similar densities. GIPSY [4] is based on fine-grained, data-oriented partitioning and on neighborhood information. It minimizes the impact of overlap by using the sparse dataset to selectively retrieve the data needed from the dense dataset, relying on connectivity information instead of a hierarchical tree traversal. By doing so, GIPSY efficiently executes a join between a sparse and a dense dataset; however, it is inefficient when joining datasets of similar density. The problem of GIPSY is that it, like other approaches, uses a static strategy and does not consider the characteristics of the datasets.

B. Motivating Application

To better understand the brain and develop new drugs for brain related diseases, the scientists in the Human Brain Project [1] build small-scale spatial models of the rat brain for brain simulations. The spatial models they design are based on millions of three-dimensional cylinders where several thousand cylinders together reconstruct the spatial shape of one neuron. To determine the locations of synapses they perform a disk-based spatial join between two types of neurons (or their corresponding cylinders), axons and dendrites. Wherever an axon intersects with a dendrite, a synapse is placed [7]. The amounts of data involved in the join make it necessary for the join to be based on disk.

Figure 3 shows the two datasets the scientists join. Axon cylinders represent 60% and dendrites 40% of the combined dataset of 250 million cylinders that model the neurons. As the illustration shows, the datasets differ significantly in data distribution: they have similar spatial extent but the axons are predominantly located at the top of the dataset. When joining theses datasets, areas of contrasting as well as similar density need to be joined efficiently, making an adaptive strategy key.

III. TRANSFORMERS Overview

As we demonstrate with the motivation experiment, each existing approach is efficient in joining a particular combination of dataset densities but none can join all combinations

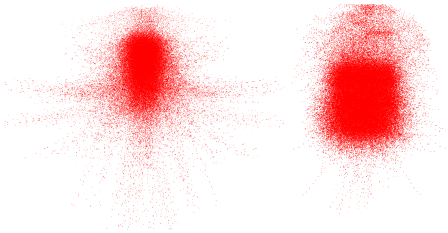

Fig. 3. Neurocience data: axons (left) and dendrites (right).

of data densities efficiently. The reason lies in their design: current approaches either use data-oriented partitioning (efficient for contrasting densities) or space-oriented partitioning (efficient for similar densities) but cannot take into account the variations of distributions and cannot adapt their join strategy.

We consequently design and use at the core of TRANSFORMERS *adaptive exploration* that robustly adapts the join strategy as well as the data layout at runtime. The join strategy is adapted by using the locally sparser data to guide the join, i.e., TRANSFORMERS uses the locally sparser data to selectively retrieve from the locally denser data only the elements needed, thereby ensuring that as little data as possible is retrieved and that as few elements as possible are tested for intersection. In case dataset A is locally sparser than dataset B, TRANSFORMERS will use the area in A to guide the join. If the roles are switched, i.e., the area in B is locally sparser, it uses B as a guide. Additionally, if the contrast in density is substantial, TRANSFORMERS splits a locally sparse area in A into finer-grained units so that each unit only needs to be joined with a small, fine-grained subset of the area in the locally denser dataset. Adapting the join strategy to the local characteristics of both datasets ensures a robust performance.

More precisely, to perform a join given two indexed datasets, TRANSFORMERS randomly picks one dataset A and uses it as the *guide* while the other is used as the *follower*. The areas $a \in A$ of the guide are visited one after the other while the connectivity information in the follower is used to navigate through it and to move to the corresponding location in the follower. The area a used for navigation is called *pivot*. Once TRANSFORMERS arrives at the location of a pivot a, it uses crawling based on the connectivity information [8], [9] to detect all spatial elements of the follower that intersect with pivot a, and then continues exploration towards a neighboring area in the guide dataset.

TRANSFORMERS adapts its strategy at runtime by switching the guide and follower: if a very sparse area is joined with a dense one, it uses the sparse area as a *guide* and the dense dataset as *follower*. Switching the roles of the datasets at runtime ensures that we can always use the locally sparser dataset to retrieve as little data as possible from the locally denser dataset, thereby robustly curbing the amount of data read and the number of comparisons.

Crucially, TRANSFORMERS also adapts the data layout on the fly: if the areas compared from both datasets have a very different number of elements, it adapts the data structure and splits the sparse pivot area into finer-grained units and joins them individually with the dense follower, retrieving only exactly the data needed. If, on the other hand, the two datasets' density is locally similar, it groups spatial elements into larger groups and joins them as a batch, thereby curbing the overhead that would result from very fine-grained partitioning, i.e., repetitive reads and comparisons. Adapting the data layout reduces the data read (and unnecessary comparisons) and thereby levels fluctuations in the join time resulting in a more robust execution time of the join.

While TRANSFORMERS borrows elements from previous approaches, i.e., data-oriented partitioning from the R-Tree, crawling from GIPSY and joining big partitions from space-oriented partitioning approaches, the departure from the state of the art lies in its ability to adapt (a) its strategy (the roles of the datasets can switch between *guide* and *follower*), and (b) the data layout on the fly thereby accomplishing robustness as well as substantially improved performance.

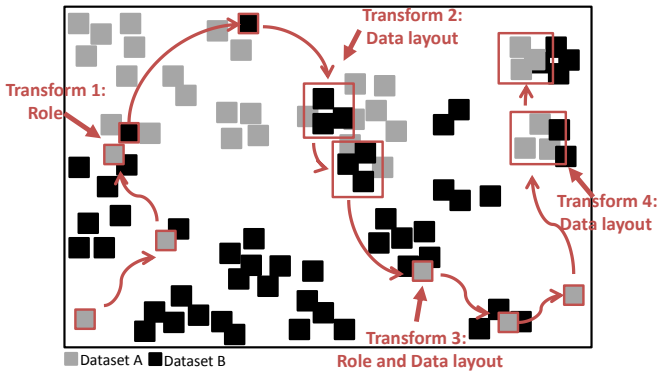

Fig. 4. TRANSFORMERS adapts to dataset characteristics.

Figure 4 illustrates how TRANSFORMERS adapts strategy and data structures to the characteristics of the datasets while performing the join; it starts with one of the datasets and uses the connectivity information to move through the dataset. Once it arrives at the position of a pivot where the follower is sparser than the guide, it switches their roles, so it joins the locally sparse with the dense area (Transform 1 in Figure 4). When it detects areas of similar local density in guide and follower it uses a coarse-grained layout (Transform 2 in Figure 4).

Clearly, instead of adapting the join strategy at runtime, we could also adapt the partitioning of datasets A as well as B and use a static join strategy. The partitioning of dataset A, however, depends on the partitioning of dataset B (and vice versa) and so the adapted partitioning can only be used to join two specific, predetermined datasets. Adapting the join strategy, on the other hand, does not depend on a particular combination and therefore enables TRANSFORMERS to reuse partitioned datasets, amortizing the overhead over several joins.

IV. TRANSFORMERS INDEXING

To enable the *adaptive exploration* TRANSFORMERS requires (a) both datasets to be partitioned and (b) connectivity information between partitions. To overcome the issues of space-oriented partitioning, TRANSFORMERS first uses fine grained data-oriented partitioning on both datasets. To enable the adaptive exploration, TRANSFORMERS further computes connectivity information between partitions, i.e., it stores for each partition a list of adjacent partitions.

Partitioning: TRANSFORMERS uses a data-oriented partitioning approach similar to STR [10] to partition the datasets. It first sorts the dataset on the x-dimension of the element center and partitions the elements along this dimension. All resulting partitions are then sorted on the y-dimension and partitioned again. The resulting partitions are sorted on the z-axis, partitioned and each partition is stored on a disk page.

The aforementioned approach for data-oriented partitioning

preserves spatial locality, i.e., spatially close elements are stored on the same disk page. Likewise, by choosing the size of the partitions at every step of the partitioning process, we can precisely determine the size of the final partitions. This (a) ensures that a partition can fit on a disk page (4K or a multiple thereof) and (b) gives us a parameter to control the granularity of the partitioning.

Data Organization: TRANSFORMERS produces two types of partitions; it first applies the partitioning algorithm on spatial elements producing *space units* and second, it groups the space units into *space nodes* using the same partitioning algorithm. The indexing phase thus produces a three-level hierarchical organization where level zero consists of space nodes, level one corresponds to the space units and level two to the individual spatial elements as illustrated in Figure 5 (left).

TRANSFORMERS stores the spatial elements on disk as space units: elements that belong to the same space unit are stored on the same disk page. It also stores meta information about each space unit in a *space descriptor*. A space descriptor summarizes a space unit *su*, i.e., it stores a pointer to the corresponding disk page, *su's* partition MBB and *su's* page MBB. The page MBB is the minimum bounding box containing all elements in a space unit (and thus on a disk page), whereas the partition MBB encloses the partition. Storing both, the page and the partition MBB, is necessary to ensure the correctness of the join process. Without the partition MBB there may be gaps between two neighboring pages MBBs (and thus the space units) in one dataset and TRANSFORMERS cannot navigate between them to explore and join the pages with the pages of the second dataset.

Finally, we group the neighboring space unit descriptors into *space nodes* that consequently store metadata information about the groups of partitions. A space node is also described with a *space descriptor*, i.e., the node's MBB that covers all its partitions and the neighbors of a space node. Figure 5 illustrates the data structures.

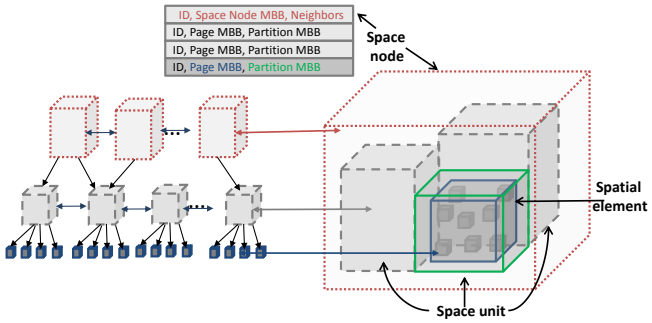

Fig. 5. The data structures: space node, space descriptor and space unit.

Connectivity: TRANSFORMERS computes the connectivity information by performing a spatial self-join on the space node MBBs, resulting in a list of all overlapping or adjacent nodes per space node. Any spatial join approach can be used for the self join. We use PBSM primarily because of its efficiency in the building phase. To decrease the amount of metadata necessary to be stored, a space unit inherits this neighborhood information from its parent space node.

V. TRANSFORMERS JOIN

Given two indexed datasets, TRANSFORMERS starts an adaptive exploration to detect intersecting pairs of spatial elements. It visits the elements of the locally sparser guide dataset, one after the other, navigating or *walking* between them using the connectivity information in the locally denser follower dataset. Once it arrives at the location of a particular guide element p, it *crawls* the neighborhood area to detect all elements of the follower that intersect with p using a *in-memory join*. Depending on the data layout used at this stage, an element p can represent either a space node, a space unit or a spatial element. TRANSFORMERS adjusts the data layout and the roles, if necessary, before the crawling step, to zoom into the area of interest.

Adaptive Walk: TRANSFORMERS first randomly assigns the roles of guide and follower to the datasets and then chooses a first pivot element, p, in the guide dataset. To determine what elements of the follower intersect with p, it needs to find a start space descriptor of the follower dataset as close as possible to p. It then uses the connectivity information to explore, i.e., recursively read all neighboring space descriptors and pick the one closest to p (smallest distance of the partition MBB to p). Exploration is repeated until a space descriptor intersecting with p is found. If no neighbor descriptor closer to p can be found (the adaptive walk is moving away from p) and the partition MBB of the closest descriptors still does not intersect with p, then p does not intersect with any element of follower. The process is illustrated in Algorithm 1.

To initially find a start space descriptor as close as possible to p (to reduce the exploration overhead), we index the Hilbert value of the center point of all space nodes in a dataset with a B+-Tree. We use B+-Trees instead of an R-Tree (or similar indexes) to avoid the issue of overlap and also to speed up building the index. To find the descriptor, TRANSFORMERS formulates a range query based on the Hilbert values of the centers of two neighboring space nodes; it only uses the B+-Tree to find the starting point of the exploration. Alternatively, the first space node of the follower dataset can be used.

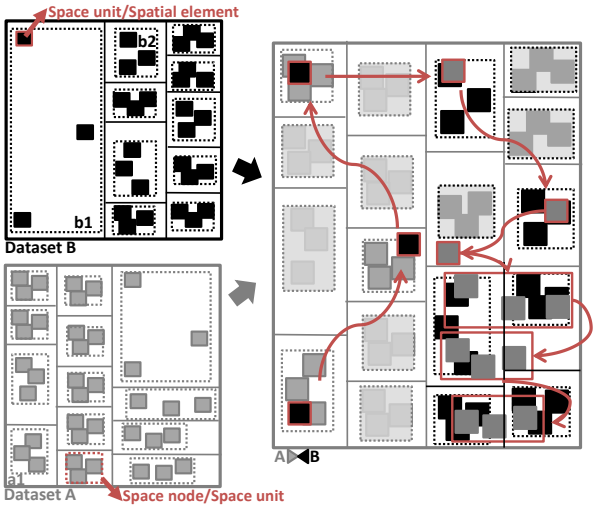

Fig. 6. Joining datasets A and B using adaptive exploration.

Adaptive Crawling: The crawl phase starts once an *intersection record* is found, i.e., a follower space descriptor whose partition MBB intersects with p. The goal of the crawl phase is to provide a *candidate set* for the final phase of the adaptive exploration process, that is, retrieving and testing actual spatial elements for intersection. Starting with the *intersection record*, similarly to the previous walk phase, the crawl phase recursively visits all neighbors until no more

Algorithm 1: Adaptive Walk Algorithm

Input: startFr: start descriptor in follower dataset
pivot: space node/unit/element
Output: clFr: closest space descriptor to pivot

$clFr$ = $startFr$
enqueue $startFr$ into $fqueue$
while $fqueue \neq \emptyset$ **do**
  dequeue follower record fr from $fqueue$
  $dist = distance(fr.partitionMBB, pivot)$
  if $dist == intersection$ **then**
    return fr
  end
  if $dist < distance(clFr.partitionMBB, pivot)$ **then**
    $clFr$ = fr
  end
  if $fqueue == \emptyset$ *AND* $!isMovingAway(clFr)$ **then**
    enqueue $clFr$'s neighbors that have not been checked in $fqueue$
  end
end
return $noIntersection$

elements intersecting with p can be found. More precisely, it starts with the *intersection record* and recursively retrieves all linked neighbor records. If a space descriptor's *page MBB* intersects with p, then its space unit page is included in the *candidate set*. On the other hand, the neighbors of a space descriptor are visited, if and only if, not only its *page MBB*, but also its *partition MBB* intersects with p. The crawl phase thus ends when no more crawl records with a partition MBB intersecting with p can be found. Then TRANSFORMERS moves to the next element in the guide dataset.

In-memory Join: Once TRANSFORMERS processes an entire space node, it joins the detected follower's *candidate set* with the *pivots* that belong to the processed space node. It partitions space in a uniform grid and assigns the elements, belonging to the *pivots*, to the cells they overlap with. Finally, it probes the grid with the elements from the *candidate set* to find pairs of intersecting elements [11]. When TRANSFORMERS uses the node level as data layout it additionally filters elements before the in-memory join. It joins the page MBBs from the guide's and follower's *candidate set* to filter out space units that do not intersect with each other, thereby reducing the data read and compared.

The pseudocode of the adaptive exploration is shown in Algorithm 2. TRANSFORMERS is finished only once all the elements from one dataset are checked for intersection. The condition $isChecked$ varies depending on the current level, i.e., for the node level we check if one dataset is fully traversed and for the unit/object level if all the elements belonging to the enclosing node/unit are checked for intersection. If at the end of the initial pass both datasets have unexamined elements the adaptive exploration process restarts taking as a guide the dataset with fewer unexamined elements. The process continues until one dataset is fully traversed, guaranteeing that all intersections are found. TRANSFORMERS collects information about all elements in the dataset during the indexing phase (space node ids). It reuses this information as a to-do list during the join process to track checked elements. Depending

Algorithm 2: Adaptive Exploration Algorithm

Input: level: current data layout
Output: intersections: result pairs
or candidateSet: input for the batch join
Data: p: space node/unit/element

while $!isChecked()$ **do**
  $p = loadCurrentPivot()$
  $intersect = adaptiveWalk(p, startRecord)$
  if $intersect == noIntersection$ **then**
    $continue$
  end
  switch $applyTransformation(p)$ **do**
    case $NoTransformation$
      $adaptiveCrawling(intersect, candidateSet)$
    case $RoleTransformation$
      $switchGuideFollower()$ & $continue$
    case $LayoutTransformation$
      $switchLayout()$
      $adaptiveExploration(level{+}{+})$
  endsw
  if $level == Node$ **then**
    $join(p, candidateSet, intersections)$
    $removeFromToDoList(p)$
  end
end
return $intersections/candidateSet$

on the current layout, we mark a space node as checked if the node itself is checked for intersection or all elements that constitute the node are checked.

Figure 6 illustrates how TRANSFORMERS uses the *elements* of the *guide* dataset to direct walking in the *follower* (for simplicity only space nodes and units are used). In this example, both datasets are grouped into partitions of three elements. TRANSFORMERS initially uses a coarse-grained layout (space nodes) and randomly chooses guide and follower dataset, e.g., A and B respectively. In *adaptiveWalk* it immediately detects an intersection between $a1$ and $b1$ and thus, before checking the actual elements for intersection and performing unnecessary reads and comparisons, it checks if it is necessary to *applyTransformation*. Considering that area $b1$ is significantly sparser compared to the same area in dataset A, TRANSFORMERS switches roles and adjusts the data layout: dataset B becomes the guide and space node $b1$ is split into space units, filtering out six partitions from dataset A. Once *adaptiveCrawling* and *join* are done, TRANSFORMERS resets the data layout to space node, keeps the dataset B as guide and uses $b2$ as next pivot, leading to additional transformations.

VI. Transformations

Crucially, when TRANSFORMERS moves to a new pivot p in the guide dataset, it adapts its strategy by adjusting the roles of guide and follower and adapts the layout.

A. Role Transformation

When joining two datasets with skewed distribution the join approach needs to adapt to the data. The roles of *pivot*, *guide* and *follower* define the configuration of TRANSFORMERS; a proper combination of roles accelerates the join and makes its performance more robust compared to static approaches.

TRANSFORMERS thus adapts the roles of *pivot*, *guide* and *follower* at runtime based on the two datasets' density ratio. Considering that both datasets rely on the same indexing strategy and thus have the same number of elements in the corresponding space units/nodes, a significant difference in volume of space units/nodes indicates that one area is sparser than the other. TRANSFORMERS thus uses the volumes enclosed by the *elements* (space node/unit) of the guide V_g and follower V_f datasets, at the location of the pivot in both datasets, to compute the ratio V_g/V_f. If the ratio is smaller than a threshold t it first switches the roles, i.e., the guide becomes the follower and the follower the guide, and then also changes the pivot (picks the element in the new guide closest to the old pivot). By adapting the roles, TRANSFORMERS ensures that it can always use the sparser dataset as guide and thus only retrieves the data needed from the denser dataset. This decision is followed by data layout transformation.

B. Data Layout Transformation

TRANSFORMERS also adapts the data layout at runtime to further reduce data read and comparisons performed. It initially designates a space node as the first pivot, i.e., it uses a coarse-grained data layout in both datasets. During the join process it may split the space node into finer-grained data structures, as a coarse-grained data layout is not a good strategy for adaptive exploration when joining areas of divergent densities. That is, in case the pivot is sparser than the corresponding area in the follower, it moves the pivot down to a level of granularity which allows for better filtering in both datasets. At runtime, TRANSFORMERS tests the ratio V_g/V_f between the datasets and changes the data layout if it is above threshold t. This decision is potentially preceded by a role switch between a guide and follower dataset if we detect a locally sparser area in the follower.

TRANSFORMERS moves seamlessly between three data layouts, predefined/produced during the indexing phase, that correspond to different levels of hierarchy as shown in Figure 7. It initially uses a coarse-grained data layout in both datasets and therefore performs adaptive exploration on level 0. In case of local density difference, it moves to a finer granularity by splitting space nodes into space units and thus performs adaptive exploration on level 1, in both datasets. Furthermore, if it detects substantial local density difference on a space unit level, it switches to the finest-grained data layout: it splits a space unit into its spatial elements, thus using a spatial element as pivot (level 2) while using the space unit as a level of granularity for the follower (level 1). TRANSFORMERS does not split the follower to object granularity as keeping track of connectivity information at this level causes significant overhead. Crucial for the data layout transformation is to transform pivot to finer granularity. We also transform the follower, when possible, to allow for better data filtering and skew detection. TRANSFORMERS decides what the data structure element e is, i.e., space node, space unit or element, based on the pivot. The same data structure is used until the end of the exploration phase, i.e., until a new pivot is chosen.

The number of levels TRANSFORMERS uses and their granularity is primarily driven by its design for use on disk. To optimize access on disk, we align data structures for disk and ensure they are page-aligned. Two levels are given: Level 2 is given by the single elements which cannot be further split as they are the smallest spatial primitive used. A second level is defined by the actual storage structure on disk: to optimize access to disk, we pack as many elements into a space unit as can fit on a disk page giving us level 1. Finally, we add a third level (level 0) that summarizes several space units on level 1 into space nodes. Level 0 is again designed page aligned, i.e., as many level 1 space units as can be summarized and stored on a disk page are combined into level 0 nodes.

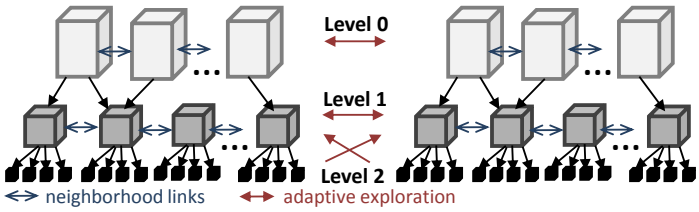

Fig. 7. The hierarchical organization of TRANSFORMERS.

TRANSFORMERS can introduce more levels between the existing ones or use more levels to recursively summarize level 0 (e.g., level -1 etc.). The former, introducing levels between existing ones, however, is inefficient as the resulting data structures would no longer be page-aligned, therefore retrieving unnecessary data (or half empty pages). Recursively summarizing level 0 leads to space nodes on higher levels which have a considerable spatial extent that soon makes the spatial extent of higher levels nodes indistinguishable from space-oriented partitioning (thereby also inheriting the same performance issues).

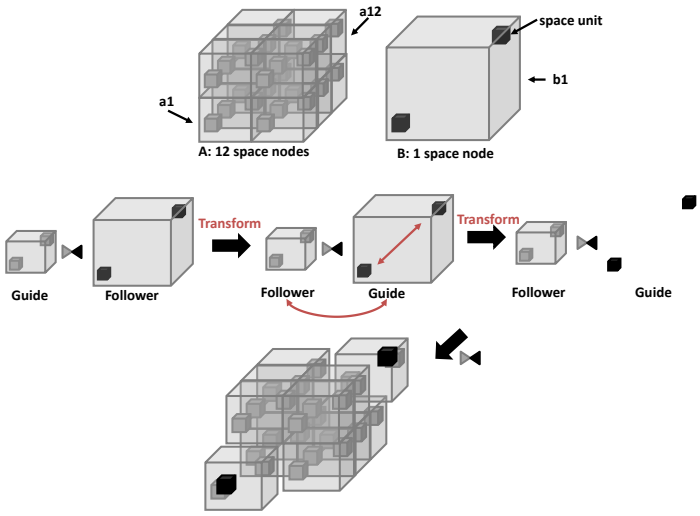

Fig. 8. Role and Data layout transformation.

Figure 8 illustrates the result of the partitioning where all space nodes contain two space units. For the sake of simplicity we do not illustrate spatial elements. Dataset A represents a densely populated area that fits 12 space nodes in the same space while B is sparsely populated containing one space node. According to the default adaptive exploration strategy, the space node *a1* will be used as a pivot to check the corresponding area (space node *b1*) of the dataset B. Without data layout transformation, the next pivot would be *a2* and eventually we would load and test all space units/elements in *a1*-*a12* against *b1*. This is unnecessary considering that *b1* overlaps only with *a1* and *a12*. A better exploration strategy in this case is to execute the exploration directed by the space units in the sparse area. TRANSFORMERS therefore switches the roles and splits the pivot, moving to a finer

granularity. It then uses dataset B as a guide and a spatial unit as pivot. By doing so it decreases the number of disk accesses and comparisons of spatial elements. At the same time, the overhead in the number of metadata comparisons is increased. Metadata comparisons, however, are not expensive as we will show in the experiments.

C. Transformation Thresholds

Setting the thresholds for transformations is important for TRANSFORMERS' performance. In particular, we need to determine the thresholds for changing the data layout, from space node to space unit and from space units to single spatial elements, and the threshold for the role transformation.

Data Transformations Threshold: The first threshold we need to determine is when TRANSFORMERS has to switch from a coarse-grained granularity (space node) to a finer-grained granularity (space unit). As described in Section VI-A, we compare the corresponding volumes in the guide and follower datasets and if the ratio exceeds a threshold t_{su} ($V_g/V_f \geq t_{su}$) we split (transform data layouts). To determine the threshold t_{su}, we first define the cost and benefit of the splitting operation.

Equation 1 defines the additional cost as the adaptive exploration (splitting means more elements to traverse), i.e., the number of new space units (nSU), after splitting a space node, times the cost of traversal and exploration (T_{ae}).

$$nSU \times T_{ae} \tag{1}$$

The average benefit of splitting, on the other hand, is essentially time saved by reading fewer space units ($nSU \times T_{io}$) and testing fewer spatial elements for intersection ($nSU \times nSO \times T_{comp}$, where nSO is the number of spatial elements in a space unit). How many space units do not need to be considered is difficult to define a priori. The ratio V_g/V_f corresponds to the maximum number of space units that can be filtered out. We adjust this value using the parameter c_{flt} = $(0, 1)$, determined at runtime based on the actual percentage of filtered elements. Equation 2 formalizes the benefit of splitting.

$$\frac{V_g}{V_f} \times c_{flt} \times nSU \times (T_{io} + nSO \times T_{comp}) \tag{2}$$

Clearly, if the benefit exceeds the cost, then we should split the space (Equation 3). Equation 4 therefore defines the corresponding threshold t_{su}. T_{ae}, T_{io} and T_{comp} are all parameters that heavily depend on the hardware of the system and are therefore best determined at runtime. TRANSFORMERS initially uses the default threshold values (Section VII-D2) that are updated after the first transformation.

$$\frac{V_g}{V_f} \geq \frac{T_{ae}}{c_{flt} \times (T_{io} + nSO \times T_{comp})}; \frac{V_g}{V_f} \geq t_{su} \tag{3}$$

$$t_{su} = \frac{T_{ae}}{c_{flt} \times (T_{io} + nSO \times T_{comp})} \tag{4}$$

Role Transformations Threshold: The data layout transformation is potentially preceded by a role switch between the guide and the follower dataset if we detect that a locally sparser area belongs to the follower dataset, that is:

$$\frac{V_f}{V_g} \geq t_{su}; \frac{V_g}{V_f} \leq \frac{1}{t_{su}}; \frac{V_g}{V_f} \leq t_{suRole}; t_{suRole} = \frac{1}{t_{su}} \tag{5}$$

Finest-grained Data Transformations Threshold: Once we are on a space unit level we can additionally adapt the data layout if we detect "extreme skew", i.e., a considerable V_g/V_f ratio. Similarly to deciding whether to split a space node into a space unit, we also need to decide if we split a space unit into single spatial elements. The reasoning behind cost and benefit is the same, except that we replace nSU with nSO (the number of spatial elements in a space unit) as Equation 6 and Equation 7 show. The term describing the cost of comparisons $nSO \times T_{comp}$ remains the same because they already depend on the number of spatial elements in a space unit.

$$nSO \times T_{ae} \tag{6}$$

$$\frac{V_g}{V_f} \times c_{flt} \times nSU \times (T_{io} + nSO \times T_{comp}) \tag{7}$$

The threshold t_{so} for deciding on splitting further thus is the ratio between the new cost and new benefit (Equation 8).

$$t_{so} = \frac{nSO \times T_{ae}}{nSU \times c_{flt} \times (T_{io} + nSO \times T_{comp})} \tag{8}$$

VII. Experimental Evaluation

In this section we describe the experimental setup & methodology, compare TRANSFORMERS against state-of-the-art spatial join approaches and then analyze its performance. To study the impact of different dataset characteristics on the performance of TRANSFORMERS we use synthetic datasets where we control the number, size and distribution of the elements. As a final test we use neuroscience datasets to compare performance on a real workload.

A. Experimental Setup

Hardware: We run the experiments on Red Hat 6.3 machines equipped with 2 quad CPUs AMD Opteron, 64-bit @ 2700 MHz, 32 GB RAM and 4 SAS disks of 300GB (10000 RPM) capacity as storage. We only use one of the disks for the experiments, i.e., no RAID configuration is used.

Software: All algorithms are implemented single-threaded in C++ for a fair comparison.

Setting: We experimentally compare TRANSFORMERS against the latest or most broadly used spatial joins, i.e., the Partition Based Spatial Merged Join (PBSM), the Synchronized R-Tree Traversal (R-TREE), and GIPSY. Like the approaches we compare it with and driven by our motivating application, TRANSFORMERS is designed to join two static spatial datasets and we do not compare it to self-joins or trajectory joins. PBSM and TRANSFORMERS use the grid hash join [11] as the in-memory join algorithm, while R-TREE uses the plane sweep. R-TREE is based on R-Trees bulkloaded using the STR approach [10]. While more sophisticated approaches can outperform STR for certain dataset characteristics (e.g., TGS [12] and PR-Tree [13]), they incur considerable overhead for partitioning the data. In practice STR balances the overhead of partitioning the data and the size of MBBs (and thus the overlap) well.

Given the absence of heuristics, we set the configuration of all approaches other than TRANSFORMERS for the best performance identified with a parameter sweep. For PBSM the configurations with 10^3 (uniform and clustered distribution) and 20^3 (neuroscience data) partitions balances the number of elements needed to be compared by the grid hash join algorithm and the number of elements replicated, deduplicated, additionally written/read to/from disk best, and therefore executes the fastest. The synchronized R-Tree approach (R-TREE) uses a fanout of 135 (based on disk page size). The parameters of TRANSFORMERS are set according to Section VI-C.

We set the disk page size to 8KB for all approaches. For all experiments we assume cold system caches and we therefore clear OS caches and disk buffers before each experiment.

B. Experimental Methodology

Synthetic Datasets: We create synthetic datasets by distributing spatial boxes in a space of 1000 units in each dimension of a three-dimensional space. The length of each side of each box is determined uniform randomly between 0 and 1. The spatial elements are distributed using a particular data distribution. We use two basic data distributions - clustered and uniform.

We use three different types of clustered datasets which differ in number and size of the clusters. For the *DenseCluster* we produce on average 700 densely populated clusters. *UniformCluster* datasets contain 100 clusters whose elements are distributed in a wide area resulting in a nearly uniform distribution while *MassiveCluster* datasets contain 5 densely populated clusters each with a fixed number (100K) of uniformly distributed elements. *DenseCluster* and *UniformCluster* use a normal distribution (μ = 500, σ = 220) to determine the centers of the clusters. Figure 9 illustrates the datasets.

The number of spatial elements in the datasets ranges between 100M and 1300M (50M-650M per dataset) resulting in a size on disk between 4.6GB and 58.2GB.

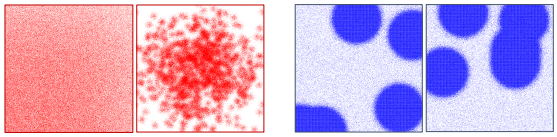

Fig. 9. *UniformCluster* & *DenseCluster* (left) and *MassiveCluster* (right) dataset samples.

Neuroscience Datasets: To evaluate TRANSFORMERS on real data we use a small part of the rat brain model represented with 450 million cylinders as elements. We take from this model a contiguous subset with a volume of 285 μm^3 and approximate the cylinders with minimum bounding boxes. In the spatial join process axons are represented by one dataset, dendrites by another and the detected intersections represent the synapses. The number of spatial elements joined ranges between 100M to 500M (50M-250M per dataset). The size on disk ranges from 5GB to 16GB.

Approach: Spatial joins typically involve two steps: filtering followed by refinement. The filtering step finds pairs of spatial elements whose approximations (MBBs) intersect with each other, while the refinement step detects the intersection between the actual shape of the elements. Considering these two steps are independent in terms of their implementation and the refinement step is application specific, we focus on the filtering like most spatial join methods [14] and like other evaluations we do not account for the refinement step.

C. Comparative Analysis

We evaluate the performance of TRANSFORMERS and compare it with other approaches (PBSM, R-TREE, GIPSY) in four sets of experiments. We first expand the motivation experiment with TRANSFORMERS demonstrating its robustness on uniform datasets. We then evaluate the performance of spatial joins on datasets with non-uniform data distributions, on uniform distributions and finally demonstrate TRANSFORMERS benefits on real neuroscience data. For the latter experiments we measure the time to index, a breakdown of the join time and the major factor of the join time, the number of intersection tests between spatial elements.

1) Robustness: This set of experiments illustrates TRANSFORMERS' robustness with respect to varying relative density. Figure 10 illustrates the performance of TRANSFORMERS when performing the set of experiments from Section II-A. The results of the join, excluding the index building time, are shown in Figure 10. The values above the curves indicate the density ratio of the datasets. TRANSFORMERS outperforms GIPSY when joining datasets with the highest density ratio (point 1000x) with a speedup of 5, while its speedup over PBSM is 6.7 when joining two dense datasets (point 1x). Its average speedup over the R-TREE is 10.

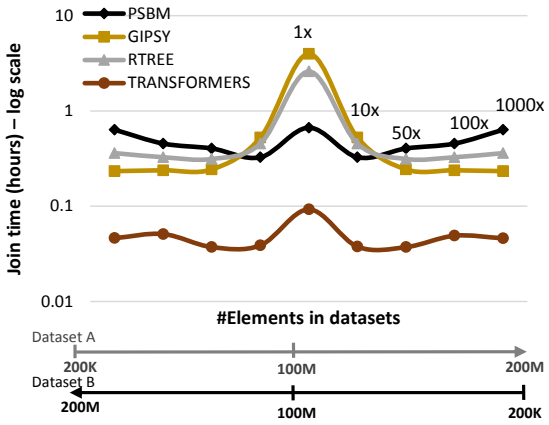

Fig. 10. Joining datasets with variable relative density.

TRANSFORMERS combines different levels of granularity and switches to the finest level only when the walking overhead is low. GIPSY's performance, on the other hand, suffers from the overhead of the directed walk on the spatial element level, which is its only level of granularity. The finest grained transformations are unnecessary when joining two densely populated datasets with a uniform distribution (point 1x). In this case, by being able to combine coarse (space node) and fine-grained granularity (space unit), TRANSFORMERS compares less data than PBSM. The performance of PBSM is also significantly affected by random reads: PBSM writes pages to disk arbitrarily while indexing (when the number of elements buffered for a cell exceeds the disk page size) leading to random reads when retrieving all elements in one cell.

This experiment exemplary demonstrates the robustness of TRANSFORMERS: by adapting to dataset characteristics at runtime (changing role and data layout), the join algorithm can compensate for extremely contrasting dataset densities.

2) Non-uniform Data Distributions: In the following set of experiments we compare and analyse TRANSFORMERS' performance on datasets with non-uniform distribution. We join synthetic datasets with clustered distribution: one dataset corresponding to *DenseCluster* and one to *UniformCluster*. We increase the size of the datasets from 350M to 650M elements, in steps of 100M and measure join and indexing time. Due to the long execution time when joining densely populated datasets, we exclude GIPSY from all these experiments and R-TREE when joining the biggest datasets (650M elements).

Indexing: The index building time is the time necessary to build the initial data structures. For PBSM this process involves creating partitions and assigning elements to them and for TRANSFORMERS partitioning the data, organizing metadata

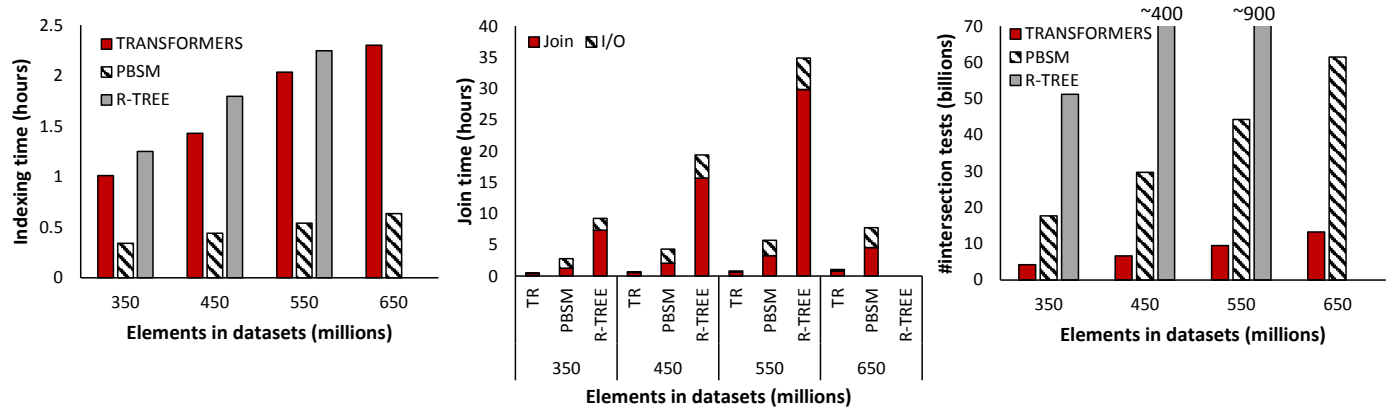

Fig. 11. Execution time breakdown and number of intersection tests for the join phase on synthetic data.

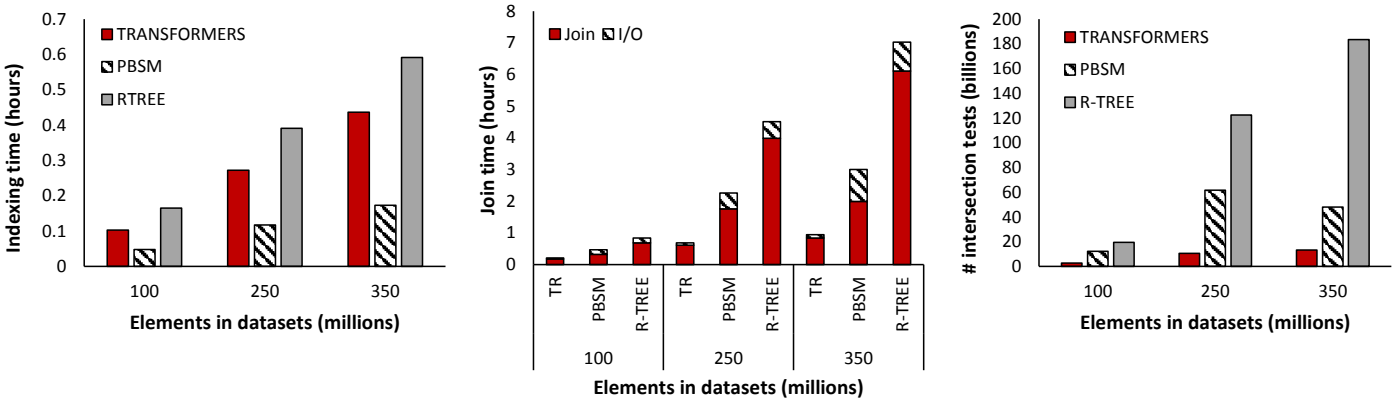

Fig. 12. Execution time breakdown and number of intersection tests for the join phase on neuroscience data.

information and introducing connectivity information. Similarly to TRANSFORMERS, R-TREE has to partition the space and additionally build all levels of the hierarchy.

Figure 11 shows the results of measuring the indexing time. The results illustrate the difference between indexes based on space- and data-oriented partitioning very well. As a space-oriented approach, PBSM only needs to assign each element to the cells of a uniform grid they overlap with. PBSM consequently outperforms TRANSFORMERS by a factor of between 2.9 - 3.6. As a data-oriented partitioning approach, on the other hand, TRANSFORMERS spends time on creating data partitions of equal size. It essentially needs to sort the spatial elements in three dimensions to produce partitions (that correspond to the space units). R-TREE partitions the data with a similar strategy but additionally has to recursively build levels, resulting in a higher indexing time.

While PBSM efficiently partitions the data, the partitions produced are unlikely to be reused efficiently. The resolution of the grid of PBSM is determined based on several factors (number of elements, spatial extent and distribution of elements) but crucially on the size of the elements of both datasets because the size of the grid cells needs to be chosen so that not too many elements are replicated. The partitions produced therefore depend on the characteristics of a particular combination of datasets and cannot efficiently be reused when joining with datasets that have considerably different characteristics.

As opposed to PBSM, TRANSFORMERS builds the indexes for each dataset separately and adapts the join execution to the characteristics of the two indexes. An index built on one dataset can therefore be reused when joining with any other dataset. The additional time TRANSFORMERS requires to index one dataset can therefore be amortized over additional joins with other datasets.

Join Performance: To analyse the join performance we break the execution time into I/O and join time. The I/O time is the time spent on loading data during the join process while join time is the time needed to join the data in memory, i.e., testing spatial elements for intersection (and related operations). The results of the experiments joining two datasets of the same size are shown in Figure 11 (middle). TRANSFORMERS (labeled TR) achieves the best results and outperforms PBSM by a factor between 5.5 & 7.4.

PBSM requires substantially more time for I/O because it reads a significant amount of unnecessary data during the join phase. Data-oriented partitioning in combination with adaptive exploration allows TRANSFORMERS to filter out 20% of the total data on average when joining *DenseCluster* datasets. When comparing the datasets with more distinctive local variations (e.g., *MassiveCluster*) TRANSFORMERS filters out on average 47% of the data while PBSM has to read all data. The execution time, however, is additionally determined by the join selectivity and the randomness of reads. PBSM inherently writes data for each partition to disk individually (and thus distributed) in the indexing phase, resulting in almost exclusively random reads during the join phase.

Figure 11 (right) shows the number of intersection tests between spatial elements for the same experiment. For TRANSFORMERS this time also includes metadata comparisons. PBSM compares all elements in one (possibly large) cell of

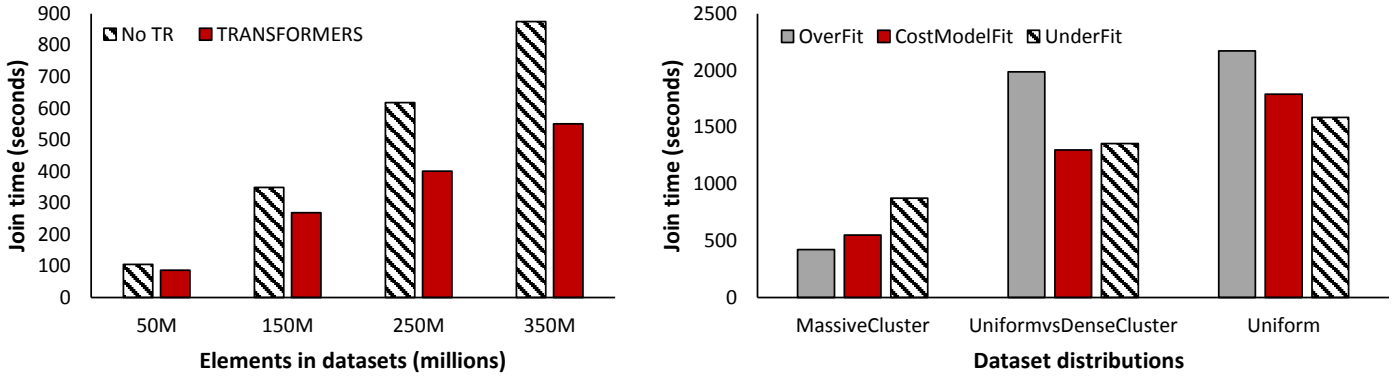

Fig. 13. Impact of transformations on join performance (left) and transformations threshold sensitivity (right).

both datasets and thus only avoids tests between elements in different cells. PBSM consequently needs to perform 4.4 times more comparisons than TRANSFORMERS which only retrieves and compares the very fine-grained partitioned data.

Given the very efficient indexing of PBSM, the overall improvement of TRANSFORMERS when taking into account the indexing and the join phase over PBSM shrinks to 2.1 - 2.5. More important, however, is the speedup in the join phase as the indexes built for TRANSFORMERS can be reused for future joins (between different datasets).

3) Uniform Data Distributions: To further demonstrate TRANSFORMERS general applicability, we join *Uniform* datasets with a uniform distribution (similar distribution and density throughout the area they cover). We vary the number of elements from 150M to 350M, in steps of 100M. The results of the join are shown in Table I.

	TRANSFORMERS	PBSM	RTREE
150M	0,16	1,02	4,55
250M	0,30	2,24	11,63
350M	0,49	4,28	24,92

TABLE I. EXECUTION TIME (HOURS) FOR DATASETS WITH UNIFORM DISTRIBUTION.

For this set of experiments TRANSFORMERS achieves a improvement of between 6.2 - 8.6 compared to PBSM. The overall improvement when joining datasets with a uniform distribution is a result of TRANSFORMERS' initial strategy that suits the datasets with similar distribution and similar number of elements. Considering that we join densely populated datasets with the uniform distribution, PBSM's performance deteriorates compared to the previous set of experiments due to the increased replication rate. By default, its strategy causes elements replication that leads to additional I/Os, comparisons and deduplication. Although the grid hash join [11] provides better performance for PBSM than the plane sweep join, it additionally increases the replication rate. The R-TREE join suffers from overlap at tree level and therefore performs on average 21 times more comparisons.

4) Neuroscience Data: To demonstrate the general applicability of TRANSFORMERS we also test its performance on neuroscience data by performing joins like the neuroscientists do. In total, at most 350 spatial elements are spatially joined, where 250M are axons and 100M are dendrites.

As the illustration shows (Figure 3) the neuroscience dataset has a skewed distribution and hence TRANSFORMERS behaves similarly as in the previous set of experiments. Figure 12 illustrates the experimental results. TRANSFORMERS achieves a speedup in join time of 2.3 - 3.3 compared to PBSM and 4.1 - 6.5 compared to R-TREE.

D. TRANSFORMERS Analysis

In the following we analyse the impact of transformations and quantify the overhead of adaptive exploration.

1) Impact of Transformations: In the following experiments we use *MassiveCluster* datasets to illustrate the impact of transformations on the performance. We join the same datasets, once with TRANSFORMERS and once with TRANSFORMERS that does not apply transformations (No TR), i.e., it just uses space nodes as the level of granularity.

With the increase of dataset size, also the data skew increases for *MassiveCluster* datasets. As the results measuring the join time in Figure 13 show, the benefit of transformations increases as the skew grows. An increase in skew triggers finer-grained transformations and thus TRANSFORMERS filters on average 47% of data resulting in an improvement in the performance between 1.2 and 1.6 compared to No TR.

2) Transformation Threshold: As discussed, TRANSFORMERS' performance depends on the transformation threshold. If the threshold is too high we will not benefit from transformations. On the other hand, if we set the transformations threshold too low the performance will be affected by the adaptive exploration overhead.

In these experiments we test the cost model using three datasets, each with 350M elements but with different data distributions: *MassiveCluster*, *UniformCluster* & *DenseCluster* and *Uniform*. To demonstrate the quality of the cost model, we use two additional configurations: *OverFit* uses 1.5 as a threshold and thus triggers many transformations and *UnderFit* uses 1,000,000 which prevents transformations, i.e., the algorithm uses default guide and follower and space nodes as data layout. All the parameters of the cost model are set and updated at runtime with an additional constraint that T_{ae} and c_{flt} are provided once the first transformation is executed. To trigger the first transformation we set the corresponding thresholds to initial values, i.e., $t_{su} = 8$, and $t_{su} = 27$. This volume ratio corresponds to the case where an edge of one MBB is two/three times bigger than the other one.

The results of the join are shown in Figure 13. When joining datasets with uniform distribution TRANSFORMERS should perform a minimal number of transformations since the two datasets do not have local variations in the distribution. The threshold proposed by the cost model leads to

performance close to *UnderFit*, the best configuration tested. *MassiveCluster* datasets have significant local variations in the distribution and therefore benefit considerably from transformations. Therefore, the threshold proposed by the cost model provides a performance very close to *OverFit* (leading to many transformations). The data distribution in *UniformCluster* & *DenseCluster* datasets (empty areas in *DenseCluster*, Figure 9) allows the coarse grained configuration to filter considerable number of elements and the performance of the cost model and *UnderFit* is thus similar.

3) Adaptive Exploration Overhead: The adaptive exploration process potentially introduces overhead. More precisely, by using a fine-grained data granularity we may lose the benefit of processing comparisons in a batch operation and we may thus unnecessarily repeat filtering operations such as distance and overlap calculations.

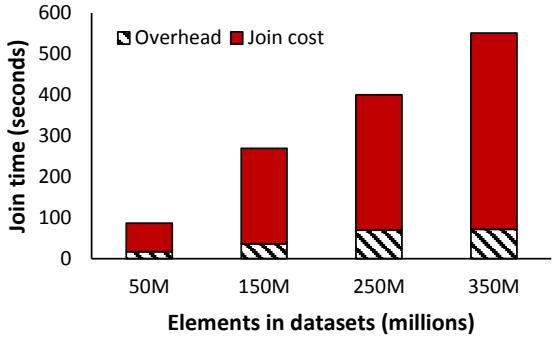

Fig. 14. Adaptive exploration overhead.

In the following experiments we measure the overhead of adaptive exploration using *MassiveCluster* datasets. To analyze the join performance, we break the execution time in join cost and adaptive exploration overhead. The join cost is the time spent on disk access and the time needed to join the data (the final *candidate set*) in memory. Everything else is considered as the overhead of adaptive exploration (Overhead).

As the results of the experiment show in Figure 14, the data layout transformations manage to keep the adaptive exploration overhead low by moving to a coarser granularity if too many elements need to be visited. On average, the adaptive exploration overhead takes 17% of the join execution time.

VIII. Related Work

Several spatial join approaches have been developed in the past with the vast majority [14] focusing on the filtering phase and only few addressing the refinement phase [15]. Because distance join approaches [16] can be trivially implemented as a variation of a spatial join (by enlarging the objects by the distance predicate) we do not distinguish between the two but instead categorize related work according to its use of data- or space-oriented partitioning.

A. Data-oriented Partitioning

Spatial join methods based on data-oriented partitioning require one or both datasets to be partitioned and indexed in a data-oriented way (e.g., an R-Tree [6]).

The synchronized R-Tree traversal [2] synchronously traverses the R-Trees [6] R_A and R_B built based on datasets A and B. Starting at the root nodes of R_A and R_B, the approach traverses the trees top down and, if two nodes $n_A \in R_A$ and $n_B \in R_B$ on the same level intersect, recursively tests their children. On the bottom level the spatial elements are tested for intersection.

The indexed nested loop join [5] only requires an index I_A on dataset A. It iterates over dataset B and queries I_A for each element $b \in B$ with b as the query. The query results are all intersections of b in A. Given the considerable cost of a query, this approach clearly is only efficient in case $A >> B$.

The seeded tree approach [17] assumes the existence of an R-Tree I_A based on dataset A and uses I_A as a template to build a second R-Tree I_B based on dataset B. Both indexes are joined with the synchronous R-Tree traversal [2] approach. As I_B is built based on I_A, the bounding boxes are aligned leading to less overlap and the synchronous join therefore has to compare fewer bounding boxes. Extensions to the basic approach use sampling to build the R-Trees faster [18] or avoid memory thrashing [19]. Sampling the spatial datasets is also used to make the spatial join interactive [20].

In case all data is known a priori, the R-Tree used can be bulkloaded to reduce overlap. Bulkloading the indexes generally reduces overlap but still cannot avoid it. Multiple approaches like STR [10], Hilbert [21], TGS [12] and the PR-Tree [13] have been developed. While TGS and PR-Tree are efficient on datasets with extreme skew and aspect ratio, Hilbert and STR perform similarly, outperforming the others on real-world data.

As a consequence of using the R-Tree as a basis, the approaches discussed so far also suffer from the same problems, namely inner node overlap and dead space. Both problems lead to a substantially increased number of disk reads as well as comparisons and hence lead to a considerably slower spatial join. Several approaches like the R+-Tree [22] or the R*-Tree [23] have been developed to mitigate the problem of overlap through replication or an improved node split algorithm. The former, however, introduces considerable overhead because duplication leads to more comparisons and disk accesses (to retrieve the duplicates). The latter reduces overlap but cannot avoid it.

The GIPSY [4] spatial join uses data-oriented partitioning but minimizes the impact of overlap by using a crawling strategy [8], [9]. GIPSY is efficient for joining a sparse dataset with a dense one. It cannot, however, efficiently join datasets of similar volumes with *arbitrary* local density variations. Besides, the performance of GIPSY relies on the ability to predetermine which dataset is dense and which one is sparse.

The JiST [24] approach goes beyond the traditional understanding of a spatial join and does not only index objects, but instead trajectories of moving objects. Based on data-oriented partitioning, JiST parametrizes trajectories on time and uses traditional indexes [6] to find the nearest trajectories to a query.

B. Space-oriented Partitioning

The class of space-oriented partitioning approaches do not partition space based on the data distribution but use uniform partitioning of the dataset space. As a consequence, each spatial element can intersect with several partitions. To address this ambiguity the assignment approaches use a multiple assignment or a multiple matching strategy.

The multiple assignment strategy assigns a copy of the element (or a reference) to each partition the element intersects with. Doing so has the advantage that only elements in the same partition need to be compared to perform the spatial join. Replicating elements, however, has several disadvantages: 1) replicated elements need more space on disk as well as

more disk reads and more comparisons for the join and 2) results may be detected twice and deduplication is required (at runtime [25] or at the end).

PBSM [3], the Partition Based Spatial Merge join, partitions the space uniformly into cells of equal size. In the first phase each element of both datasets is assigned/replicated to all cells it overlaps with. In the second phase PBSM iterates over all cells $c \in C$ and tests all elements of dataset A in c against all elements of dataset B in c to find pairwise intersections.

The multiple matching strategy avoids replication of elements and copies each element only to one partition it intersects with. Doing so, however, also means that several partitions (that share a border) potentially need to be compared with each other because an element could be assigned to one of several different partitions.

SSSJ [26], the Scalable Sweeping-Based Spatial Join, partitions space into n strips of equal width in one dimension and assigns each element e of both datasets to the strip where e is fully contained. In each of the n strips it uses a plane-sweep approach to determine all intersections between elements of datasets A and B. Elements intersecting with several strips (e.g., from strip i to strip k) are assigned to set S_{ik}. When swiping strip j all sets S_{jk} with $j < n < k$ are also joined.

The size separation spatial join (S3 [27]) uses a hierarchy of equi-width grids of increasing granularity. Each element of both datasets is assigned to the lowest level in the hierarchy where it only overlaps with one cell. To perform the join S3 iterates over each cell c in the hierarchy and joins it with all cells that cover c on a higher level.

The problem of skewed datasets has already been studied in a distributed setting [28] to investigate the impact of skew. The work done for datasets with skewed distribution, however, focuses on partitioning and distributing the datasets so that all worker nodes perform a similar amount of work, while still considering all the data. Our approach, on the other hand, deals with skew by minimizing unnecessary I/Os and comparisons.

IX. Conclusions

This paper identifies the problem of spatial join *robustness*, which arises when joining spatial datasets of similar volumes but locally varying densities. As we show, current methods cannot efficiently perform a join between such datasets, which are prevalent in applications across sciences; such methods read too much data and/or require too many comparisons.

We propose TRANSFORMERS, a method that achieves robust spatial joins by adapting to local data characteristics. TRANSFORMERS partitions the data in advance, but, contrary to previous work, does not rely solely on that partitioning; it also adapts its execution in an on the fly, data-driven manner. First, it uses the locally sparser dataset to guide data retrieval, ensuring that only strictly needed data from the locally denser dataset are retrieved. Second, it adjusts the employed data layout, ensuring that only the relevant parts of the locally denser dataset are compared.

We show that TRANSFORMERS achieves robust and efficient joins. Thanks to its adaptivity, it achieves a speedup between 2 and 8 in the join phase compared to PBSM, the fastest state-of-the-art method throughout the density ratio spectrum. Moreover, it is scalable, capable to perform on ever bigger data of growing density variations.

Acknowledgements

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n 604102 (HBP).

References

[1] H. Markram, K. Meier, S. Grillner, R. Frackowiak, S. Dehaene, A. Knoll, H. Sompolinsky, K. Verstreken, J. DeFelipe, S. Grant, and J.-P. Changeux, "Introducing the Human Brain Project," vol. 7, no. 1, 2011, fET '11.

[2] T. Brinkhoff, H.-P. Kriegel, and B. Seeger, "Efficient Processing of Spatial Joins using R-Trees," in *SIGMOD '93*.

[3] J. Patel and D. DeWitt, "Partition Based Spatial-Merge Join," in *SIGMOD '96*.

[4] M. Pavlovic, F. Tauheed, T. Heinis, and A. Ailamaki, "GIPSY: Joining Spatial Datasets with Contrasting Density," in *SSDBM*, 2013.

[5] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, 3rd ed. Addison Wesley, 2000.

[6] A. Guttman, "R-trees: a Dynamic Index Structure for Spatial Searching," in *SIGMOD '84*.

[7] J. Kozloski, K. Sfyrakis, S. Hill, F. Schurmann, C. Peck, and H. Markram, "Identifying, Tabulating, and Analyzing Contacts between Branched Neuron Morphologies," *IBM Journal of Research and Development*, 2008.

[8] F. Tauheed, L. Biveinis, T. Heinis, F. Schürmann, H. Markram, and A. Ailamaki, "Accelerating Range Queries For Brain Simulations," in *ICDE '12*.

[9] S. Papadomanolakis, A. Ailamaki, J. C. Lopez, T. Tu, D. R. O'Hallaron, and G. Heber, "Efficient Query Processing on Unstructured Tetrahedral Meshes," in *SIGMOD '06*.

[10] S. Leutenegger, M. Lopez, and J. Edgington, "STR: a Simple and Efficient Algorithm for R-tree Packing," in *ICDE '97*.

[11] F. Tauheed, T. Heinis, and A. Ailamaki, "Configuring Spatial Grids for Efficient Main Memory Joins," in *BICOD '15*.

[12] Y. J. Garcia, M. A. Lopez, and S. T. Leutenegger, "A Greedy Algorithm for Bulk Loading R-trees," in *Advances in Geographic Information Systems (AGIS '98)*.

[13] L. Arge, M. D. Berg, H. Haverkort, and K. Yi, "The Priority R-tree: A Practically Efficient and Worst-case Optimal R-tree," *ACM Transactions on Algorithms*, vol. 4, no. 1, pp. 1–30, 2008.

[14] E. H. Jacox and H. Samet, "Spatial Join Techniques," *ACM Transactions on Database Systems*, vol. 32, no. 1, p. 7, 2007.

[15] D. J. Abel, V. Gaede, R. A. Power, and X. Zhou, "Caching Strategies for Spatial Joins," vol. 3, no. 1.

[16] J. Sankaranarayanan, H. Alborzi, and H. Samet, "Distance Join Queries on Spatial Networks," in *GIS '06*.

[17] M. Lo and C. Ravishankar, "Spatial Joins Using Seeded Trees," in *SIGMOD '94*.

[18] ——, "Spatial Hash-joins," in *SIGMOD '96*.

[19] N. Mamoulis and D. Papadias, "Slot Index Spatial Join," *IEEE TKDE*, 2003.

[20] S. Alkobaisi, W. D. Bae, P. Vojtěchovský, and S. Narayanappa, "An Interactive Framework for Spatial Joins: A Statistical Approach to Data Analysis in GIS," vol. 16, no. 2.

[21] I. Kamel and C. Faloutsos, "On Packing R-trees," in *CIKM '93*.

[22] T. K. Sellis, N. Roussopoulos, and C. Faloutsos, "The R+-Tree: A Dynamic Index for Multi-Dimensional Objects," in *VLDB '87*.

[23] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: an Efficient and Robust Access Method for Points and Rectangles," in *SIGMOD '90*.

[24] Y. Chen and J. M. Patel, "Design and Evaluation of Trajectory Join Algorithms," in *GIS '09*.

[25] J.-P. Dittrich and B. Seeger, "Data Redundancy and Duplicate Detection in Spatial Join Processing," in *ICDE 2000*.

[26] L. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, and J. S. Vitter, "Scalable Sweeping-Based Spatial Join," in *VLDB '98*.

[27] N. Koudas and K. C. Sevcik, "Size Separation Spatial Join," in *SIGMOD '97*.

[28] J. Patel and D. DeWitt, "Clone Join and Shadow Join: two Parallel Spatial Join Algorithms," in *SIGSPATIAL '00*.